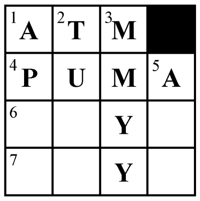
Crossword Puzzle Resolution via Monte Carlo Tree Search

Lihan Chen[1], Jingping Liu[1], Sihang Jiang[1], Chao Wang[1], Jiaqing Liang[1], Yanghua Xiao[1,3]*, Sheng Zhang[2], Rui Song[2]*

[1] Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
[2] North Carolina State University, Raleigh, US
[3] Fudan-Aishu Cognitive Intelligence Joint Research Center
lhc825@gmail.com, jpliu17@fudan.edu.cn, tedsihangjiang@gmail.com, 17110240038@fudan.edu.cn, l.j.q.light@gmail.com, shawyh@fudan.edu.cn, szhang37@ncsu.edu, rsong@ncsu.edu

Abstract

Although the development of AI in games is remarkable, intelligent machines still lag behind humans in games that require the ability of language understanding. In this paper, we focus on the crossword puzzle resolution task. Solving crossword puzzles is a challenging task since it requires the ability to understand natural language and the ability to execute a search over possible answers to find an optimal set of solutions for the grid. Previous solutions are devoted to exploiting heuristic strategies in search to find solutions while having limited ability to explore the search space. We propose a solution for crossword puzzle resolution based on Monte Carlo tree search (MCTS). As far as we know, we are the first to model the crossword puzzle resolution problem as a Markov Decision Process and apply MCTS to solve it. We construct a dataset for crossword puzzle resolution based on daily puzzles from New York Times with detailed specifications on both the puzzle and clue database selection. Our method can achieve an accuracy of 97% on the dataset.



Crossword Grid

Across	Down
1. $ dispenser	1. Pit-__
4. Adidas alternative	2. Spanish pipe
6. What the club gets without playing	3. Credit-card exp. date format
7. 1992 Robin Williams movie	5. Two-time loser to DDE

Clues

Figure 1: A toy example of an unfinished crossword puzzle, constructed by us with real clues from published crossword puzzles. In the grid, each slot is hinted by a clue and starts with a square whose top-left corner is indexed by the corresponding clue number. Words reading from left to right are hinted by the clues in the "across" column (e.g. `PUMA` at "4-across" is hinted by the clue *Adidas alternative*), and those from top to down are the "down" column (e.g. `MMYY` at "3-down" is hinted by the clue *Credit-card exp. date format*). Words end when reaching the border or a black square.

Introduction

With the remarkable development of AI, current intelligent systems surpass humans in many games requiring computational power, such as GO (Silver et al. 2017). However, in games that require ability such as language understanding and knowledge usage, intelligent machines still lag behind humans. For example, in American Crossword Puzzle Tournament 2021[1], although the AI participant Dr. Fill (Ginsberg 2011) surpasses human players in terms of speed while still having inferior performance on accuracy.

In this paper, we focus on solving crossword puzzles (CPs). A crossword puzzle usually consists of a crossword grid and a set of clues, where the former consists of white and black squares and the latter include a clue for each slot. The slot consists of consecutive white squares in a row or column. The player is required to fill words in slots based on clues, where the words share a letter at the square of the intersection. The black squares are used to separate the slots. Figure 1 presents an example of a crossword puzzle. The goal of the game is to find the best solution, where each white square is filled with a letter and every word in the slot matches the semantics of the corresponding clue best.

The challenge for solving CPs is twofold. First, it is computationally costly to search for a best solution in an immense space. Any letter sequence is a possible "word" in a CP. For example, in Figure 1, the answer to "3-down" is `MMYY`, which is not a standard word. The number of white squares in the grid of a typical crossword puzzle is around 200, the total search space thus is 26^{200} since any letter could be potentially in a square. It is computationally intractable to explore such a huge search space. Even though the clues can be used for pruning, the search space is still large due to the ambiguity and obscurity of the clues. Second, it is still challenging to find the right answer from clues. Clues help human players to resolve the puzzle. However, clues are often obscure (e.g. "*$ dispenser*"), ambiguous (e.g. "*Adidas alternative*"), and diverse (e.g. blank filling clues, knowledge-related clues). Hence, CP problem is a good task to test the ability of machines in playing games that require the understanding of natural language and the world.

There are some existing solutions for the automatic resolution of crossword puzzles, but they still have many problems. First, they have limited ability to handle the huge

*Corresponding authors.

[1] https://www.crosswordtournament.com/2021/index.htm

search space. Most of these solutions search for the best solution among a huge solution space with a greedy search heuristic (Ernandes et al. 2005; Ginsberg 2011). In general, it is tricky to design a heuristic that could balance effectiveness and efficiency. Second, they fail to provide a good evaluation for candidate solutions. The evaluation relies on the clue-word matching for each clue. Previous methods (Barlacchi et al. 2014; Moschitti et al. 2015; Severyn et al. 2015) train a ranking model to select appropriate words for clues. However, the candidate words are generated from different sources due to the diversity of clues. In general, different sources need different ranking mechanisms. As a result, any single ranking mechanism will fail to evaluate the words.

We propose a novel framework for crossword puzzle resolution. **For the problem of huge search space**, we formalize it as a Markov Decision Process (MDP) and adopt the Monte Carlo tree search (MCTS) to solve it. MCTS can asymptotically improve the strategy by iterative simulations of the word-filling procedure in the CP game. It has been proven to be effective in tasks of sequential decisions, such as games and planning problems (Browne et al. 2012). However, it is still challenging to apply MCTS to our problem. First, the action space of word-filling is extremely large because candidate words could be any possible letter sequence. Even with the constraints of clues, the action space cannot be effectively narrowed down due to the obscurity and ambiguity of the clues. We propose an action space reduction strategy to solve this problem. Second, the search procedure in MCTS (i.e., tree policy) is unstable for the CP problem. The tree policy usually relies on the estimated action value (Silver et al. 2017), which is unfortunately of high variance for different puzzles and different states (Figure 3 in the experiment section). We propose a variance control strategy to solve this problem. **For the problem of solution evaluation**, it corresponds to the reward design in the formalization of MDP, which serves as the key to guide our MCTS-based solution. To handle the candidates from different sources, we train an individual ranking model for each candidate generation source. We design a novel reward function considering multiple ranking mechanisms of words from different sources and tune it to reflect the influence of the global grid constraint. Our contribution is threefold.

- To the best of our knowledge, we are the first to apply MCTS to solve the crossword puzzle. Specifically, we propose an action space reduction strategy to optimize the search and a variance control strategy to optimize the tree policy of MCTS. We believe those techniques can be applied to other tasks with similar problems.
- We propose a novel reward function that aggregates the clue-word matching from different sources. Our aggregated reward function is effective to guide MCTS to find the solution that satisfies the grid constraint and clue constraints simultaneously.
- We construct a dataset for crossword puzzle resolution based on daily puzzles from New York Times (NYT) and our method achieves state-of-the-art performance in the dataset. Our dataset is the largest (57 puzzles for test, 85 for validation), and most complete (with fixed 4.5 million seen clues in the dataset) in public.

Overview

Problem Formalization

Each CP, in this paper, is formalized as a quadruple $\langle X, C, D, F\rangle$. $X = [x_1, ..., x_q]$ is an array of variables, where each variable $x_i (i = 1, ..., q)$ refers to a word slot to be filled (e.g. "1-across"). Each word slot x_i corresponds to a clue c_i which is a constant. $C = [c_1, ..., c_q]$ is the array of clues. D is the domain set of all possible words, which includes dictionary vocabularies, e.g., `puma`, and other strings, e.g., date format `MMYY`. F represents the grid constraints, including cross consistency (e.g. in Figure 1, the second letter of the "3-down" word `MMYY` equals to the third letter of the "4-across" word `PUMA`), and length constraint (e.g. in Figure 1, the "1-across" word is a 3-letter word).

To represent the solution, we define an assignment of X as $W = [w_1, ..., w_q]$, where each w_i is an assignment for the corresponding word slot x_i. Each w_i is a word in the domain (i.e. $w_i \in D$), or unfinished (i.e. $w_i \in \Delta$). Δ is the set containing all the unfinished words with at least one blank (i.e. an unfinished grid is also an assignment). For example, the "6-across" of the assignment in the grid of Figure 1 is an unfinished word with 3 blanks and belongs to Δ. The constraint F is formulated as an indicator function of W: if all the words in W satisfy the grid constraints F, $F(W) = 1$; otherwise, $F(W) = 0$. CP problem is thus formulated as:

$$W = [w_1, ..., w_q] \text{ s.t. } \begin{cases} w_i \in D, \forall i \\ F(W) = 1 \\ w_i \text{ satisfies } c_i, \forall i \end{cases} . \tag{1}$$

The first is the completeness constraint. The second is the grid constraint F. The last clue constraint requires the words in W should semantically match with the associated clues in C. Since the clue constraint is not black-or-white, we use a function $r(w_i, c_i) \in \mathbb{R}, w_i \in D \cup \Delta, c_i \in C$ to measure the goodness of each clue-word pair. A large $r(w_i, c_i)$ means w_i is highly matched with c_i. Since the above constraints are independent of each other, our goal is to find an assignment maximizing the sum of rewards for all clues:

$$\arg\max_{W} \sum_{1 \le i \le q} r(w_i, c_i) \quad s.t. \;\; w_i \in D, \forall i, F(W) = 1. \tag{2}$$

As a sequential decision-making problem, the solving of a CP can be naturally formalized as a Markov Decision Process (MDP). An MDP has mainly four components, including the set of *states* S, the set of *actions* A, *transition function* $T(s, a) \in S$, and *reward* $R(s) \in \mathbb{R}$, where $a \in A, s \in S$. In a CP game, each state $s = [s_1, s_2, ..., s_q] \in S$ corresponds to an assignment W that satisfies the grid constraint, i.e. $F(W) = 1$. For convenience, we use s_i to denote w_i when no conflict is introduced. An action $a = \{(i, v) | 1 \le i \le q, v \in D\} \in A$ includes two components: selecting a word slot x_i and filling it with a word v. We define the action space A_s for state s as the actions that satisfy the current grid constraint of s. The state s is *terminal* if $A_s = \emptyset$. The deterministic mapping from a state-action pair to another state is

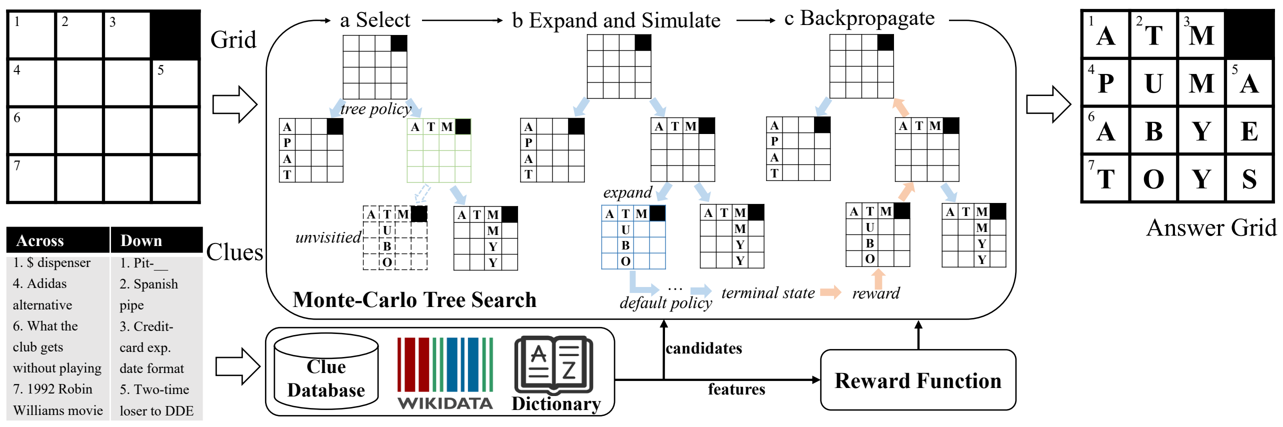


Figure 2: The framework of our crossword puzzle resolution system based on MCTS. The algorithm iteratively executes three steps to build a search tree and update the best solution.

denoted as the transition function $T(s, a)$, and the *cumulative reward* $R(s)$ for each state s is computed by the average score of r.

$$R(s) = \frac{1}{q} \sum_{1 \leq i \leq q} r(s_i, c_i). \tag{3}$$

Our Solution

In the CP problem, each puzzle with its specific clues and grid constraints corresponds to an independent MDP. A typical solution for an MDP is to obtain an optimal *policy* that iteratively takes the action that maximizes the expected cumulative reward. The terminal state produced by the optimal policy with maximum cumulative reward corresponds to the answer to our CP problem. To acquire such a policy, a search must be performed on all the states while the state space is too huge to perform a complete exploration. Thus, as shown in Figure 2, we adopt MCTS, incorporating the strategy of random sampling into the construction of the policy, to solve the MDP of CP. Based on MCTS, our algorithm asymptotically refines the policy and finds the solution by iteratively simulating the CP game. However, it is still challenging to apply MCTS in the CP problem. First, the action space of word-filling is extremely large. We need to utilize knowledge from multiple data sources to narrow down the action space based on the clues while avoiding omitting at the same time. Second, the reward is not given as a clear signal. We need to carefully design the reward function based on the features from those data sources. In the following sections, we first introduce the detailed algorithm for CP, then illustrate the specific design of the reward function.

Crossword Puzzle Resolution Based on MCTS

In this section, we first describe the overall algorithm; then, we elaborate on our designs for specific problems in CP.

Algorithm Overview

Our algorithm iteratively simulates the game and updates the best solution until a time limit is reached. During the iteration, a search tree with visited nodes from the past simulation is gradually built, and the policy based on the search tree becomes more and more accurate. Each node of the tree corresponds to a state and holds two values, the number $N(\cdot)$ of times it has been visited and a value $G(\cdot)$ that corresponds to the total returns (cumulative rewards from current states to terminal states) of all playouts that passed through this state. The details are illustrated in Algorithm 1. Each iteration consists of the following four steps.

Selection (line 5-9). Starting at the root node, a tree policy is recursively applied to select children until an expandable node is encountered. A node is expandable if it has unvisited children. The tree policy is to select an action maximizing an *upper confidence bound* (UCB) (Browne et al. 2012; Silver et al. 2017) to trade off the actions with high estimated cumulative rewards against those less-visited ones. For the calculation of UCB in the tree policy, we adopt the widely-used P-UCT algorithm (Silver et al. 2017; Rosin 2011), $\text{UCB}(s, a) = Q(s, a) + \lambda\sqrt{N(s)}P(s, a)/(1 + N(s'))$, where λ is a hyper-parameter used to determine the degree of exploration and $P(s, a)$ is the prior probability of taking an action a on s.This search control strategy initially prefers actions with high prior probability and low visit count, but asymptotically prefers actions with high action value.

Expansion and simulation (lines 10-15). The expansion step starts at an expandable node with state s and adds an unvisited child of the node to the search tree. If there are multiple unvisited children, we choose the child according to the prior probability P. The simulation step reaches a terminal state s_t from the expanded child by iteratively applying the default policy and obtains a cumulative reward $R(s_t)$.

Backpropagation (line 16-19). In the backpropagation step, we update the statistics of nodes $(G(s), N(s))$ along the path from the expanded node to the root with $R(s_t)$.

Update (line 20-22). After the three steps above, the search tree is updated and a potential solution corresponding to s_t is obtained. We execute a postprocessing procedure $Post$ (line 20) to s_t and update the best solution (line 21-22). We adopt the postprocessing procedure from Dr. Fill (Ginsberg 2011), which has been proven to be effective for the algorithm to fix a small number of letters filled wrong.

In our CP problem, it is common that different sequences

Algorithm 1: Monte Carlo tree search for CP

```
Input: CP $\langle X, C, D, F\rangle$, reward $R$, prior probability $P$,
       transition $T$, and postprocessing function $Post$
Output: A state of largest rewards $bestS$
1  $maxR \leftarrow -\infty; bestS \leftarrow null$
2  $G \leftarrow \{s_0 : 0\}; N \leftarrow \{s_0 : 0\}$
3  while within time limit do
4      $s \leftarrow s_0$
5      $E \leftarrow \{a : a \in A_s \ \& \ T(s,a) \notin G\}$
6      while $E = \emptyset$ do
7          $a \leftarrow \arg\max_{a \in A_s} \text{UCB}(s,a)$
8          $s \leftarrow T(s,a)$
9          $E \leftarrow \{a : a \in A_s \ \& \ T(s,a) \notin G\}$
10     $a \leftarrow \arg\max_{a \in E} P(s,a)$
11     $s \leftarrow T(s,a); s_t \leftarrow s$
12     while $s_t$ is not terminal do
13         Sample $a \in A_{s_t}$ with $P(s_t,a)$
14         $s_t \leftarrow T(s_t,a)$
15     $G(s) \leftarrow 0; N(s) \leftarrow 0$
16     while $s$ is not null do
17         $N(s) \leftarrow N(s) + 1$
18         $G(s) \leftarrow G(s) + R(s_t) - R(s)$
19         $s \leftarrow Parent(s)$
20     $s_t \leftarrow Post(s_t)$
21     if $R(s_t) > maxR$ then
22         $maxR \leftarrow R(s_t); bestS \leftarrow s_t$
23 return $bestS$
```

of actions may lead to the same assignment. There are many cases where different nodes in the search tree correspond to the same state. In a deterministic MDP, a standard MCTS stores statistics to nodes (or edges), which means the statistics of a state may be distributed over multiple nodes corresponding to it. The statistics for the corresponding state are not fully utilized when estimating the expected rewards of a node. Thus, we aggregate the statistics of nodes corresponding to the same state. It enhances the tree policy by reducing the estimation bias of the expected reward of action a on the state s. With this strategy, the search tree can be regarded as a directed acyclic graph (DAG). Similar strategies are also introduced as Monte Carlo Graph Search (MCGS) in some recent works (Leurent and Maillard 2020; Czech et al. 2021). We still refer to it as a variant of MCTS in this paper because they share the same procedure.

Variance Control for UCB

In the P-UCT algorithm, the action value Q can be easily estimated by statistics in the tree, $Q(s,a) = G(s')/N(s')$, where $s' = T(s,a)$ is the next state. However, the estimated Q is of high variance due to various puzzles (obscure clues may get lower rewards) and different states (successive states have lower Q-value). Thus, we propose to use the state value of s, i.e. $V(s) = G(s)/N(s)$, as normalization for this term. The UCB in this paper is defined as

$$\text{UCB}(s,a) = \frac{G(s')N(s)}{G(s)N(s')} + \lambda \frac{\sqrt{N(s)}P(s,a)}{1+N(s')}. \quad (4)$$

The first term can be viewed as the advantage of the action. Although the range of Q in UCT algorithms should be in $[0,1]$ (Meyer and Gurevych 2012), our strategy works well in practice by tuning a proper value for hyper-parameter λ. This strategy shares similar ideas of setting λ to state or action values (Balla and Fern 2009; Bonet and Geffner 2012; Keller and Helmert 2013). The key distinction here is that we present the variance control as a normalization mechanism, thus we can further improve the performance by tuning λ.

Action Space Reduction

In the expansion and simulation step, the algorithm will explore all the possible actions for each state to fully expand a node. However, the action space is huge since an action needs to consider every word v in the immense domain D for each word slot. To solve this problem, we propose an action space reduction strategy, where we consider probable candidates first and leave the others for later. First, we respectively rank candidate words v for each clue c_i with the reward function $r(v, c_i)$. Then, we only include the filling of the rank-1 word for each clue in the action space. Last, for a non-terminal state s, we design an *additional action* to take other words into consideration.

$$A_s = \{(i, \arg\max_v r(v, c_i)) : i \in [1,q] \wedge s_i \in \Delta\} \cup \{\alpha\}. \quad (5)$$

The additional action α makes a special transition from the current state to another state with the same assignment. The new state eliminates the rank-1 candidate for each clue and will consider filling the rest of the candidates as actions. Note that the additional action is performed recursively, so it theoretically takes all the candidate words for each clue into consideration. When the rank-1 candidates are eliminated, the rank-2 candidates become rank-1 in the new state. The new state is also associated with an additional action eliminating the current rank-1 candidates. With this design, the definition of the state is technically changed to the assignment with the number of top-ranked candidates eliminated.

Candidate Generation

The MCTS algorithm needs to explore every possible action for a state to expand. Even though the subtle strategy is adopted to reduce the action space, it is still time-consuming to compute the reward function $r(v, c_i)$ for each word $v \in D$ of each slot x_i.Hence, it is necessary to generate candidate words with high quality, which in turn will be explored first in the MCTS algorithm. We divide the candidate generation modules into two categories, *clue-dependent generation* (CDG), and *clue-independent generation* (CIG). The CDG module generates candidates before the search, and the CIG module generates candidates during the search when no candidate from CDG satisfies the grid constraints.

In the CDG, we query the puzzle clue to retrieve top-ranked clues from the database via BM25 score (Robertson and Zaragoza 2009) and return the corresponding γ_1 answers with the proper length. After that, we re-rank these candidates with score $S_{seen}(v, c_i)$ produced by a neural network based on BERT (Devlin et al. 2018). To improve the recall, CIG is further used to retrieve less reliable but comprehensive candidates when none of the candidates could be used from CDG. CIG first looks up a valid candidate word in the vocabulary. We use a binary score $S_{vocab}(v)$ to indicate whether the word (string) $v \in D$ is in the vocabulary (1) or not (0). Since this source is less reliable, this module returns candidates only when the grid constraints are so strong that no more than γ_2 candidates in the vocabulary can be filled. If no word is found in the vocabulary, the CIG tries to generate a multi-word compound phrase as the candidate with grid constraint via a segmentor (Littman et al. 2002) which segments the grid constraint on this slot with the maximum logarithm probability $S_{phrase}(v)$, produced by the unigram language model.

Default Policy

The default policy is to use prior probability distribution to take an action from the action space for each state in the simulation step. As mentioned before, the action space A_s of the state s includes the actions that fill the rank-1 candidate word to each word slot and the additional action. Inspired by (Ginsberg 2011), a good strategy to fill the word is to maximize the difference between the first largest and the second-largest score. Thus, we first define the score for a prior heuristic strategy for an action $a \in A_s$ as

$$H(s,a) = \begin{cases} r(v, c_i) - \max2_{v'} r(v', c_i) & \text{if } a = (i, v) \\ 0 & \text{otherwise} \end{cases}, \quad (6)$$

where $\max2(\cdot)$ means the second-largest score.

A normalized function is then used to convert the function H into a probability distribution $P(s,a)$, which is defined as

$$P(s,a) = \frac{exp(H(s,a)/\tau)}{\sum_{a' \in A_s} exp(H(s,a')/\tau)}, \quad (7)$$

where τ is temperature hyper-parameter. When τ gets smaller, the default policy tends to exploit the actions with a high value of H, and when τ gets larger, the default policy tends to explore the actions with a low value of H.

Reward Function

Unlike games with clear reward signals (e.g. win or lose in GO), the outcome of CP needs careful evaluation. The goodness of the final solution depends on the candidate word matching scores r for individual clues. It is intuitive to train a clue-word matching model for individual clues with semantic features and learning-to-rank techniques (Barlacchi et al. 2014; Severyn et al. 2015). However, this solution cannot guarantee optimal evaluation for the solution because one ranking model cannot handle the candidates from different sources. In this section, we first introduce the features we used; after that, we describe a ranking model baseline; last, we illustrate our design.

Features

The essence of the reward function $r(s_i, c_i)$ is evaluated by the matching degree between a word and a clue. We introduce features that are able to reflect how well a word matches a clue, in Table 1. The features are divided into two categories, clue-aware features, which measure the matching degree of the clue and the word, and clue-free features,

Type	Feature	Description
Clue-aware	S_{seen}	Score of seen clue retrieval
	W2V	Word2vec similarity of word and clue
	POS	POS tag matching of clue and word
Clue-free	Count	Log # of occurrences in clue database
	Unigram	Log unigram counts
	S_{vocab}	Whether the word is in the vocabulary
	S_{phrase}	Score of phrase-based generation

Table 1: Specifications of features in the reward function.

which measure whether the word itself is good without the clue. The clue-aware features are effective when the clue is explicit or easy to understand, while the clue-free features work when the clue is obscure. Aside from those coming from the retrieval scores, we adopt several additional features. For clue-free features, we consider the number of occurrences in the clue database and the unigram count of the word. Those features are helpful for selecting the answer when the signal from the clue is vague. For clue-aware features, we additionally consider POS tag matching degree and word2vec (Mikolov et al. 2013) similarity.

A Ranking Model Baseline

A typical solution is to use the scores provided by a ranking model trained on the ground-truth clue-answer pairs as the reward function $r(w,c), w \in D, c \in C$ for answers to a single clue (Moschitti et al. 2015). We utilize the features above to train a ranking model $f(w,c;\theta)$ for how well a candidate word w matches the clue c. We adopt the ListMLE (Xia et al. 2008) algorithm for the optimization of our ranking model. Specifically, we use the following loss function to optimize the model.

$$L(f;\theta) = -\sum_{c} \sum_{w \in W_c} y_c(w) \log \frac{exp(f(w,c;\theta))}{\sum_{w' \in W_c} \exp(f(w',c;\theta))}. \quad (8)$$

W_c is the candidate answer set for clue c and y_c represents the ground truth. Specifically, $y_c(w) = 1$ if w is the answer of c; $y_c(w) = 0$, otherwise. Then we define the reward function as the trained ranking model, i.e. $r(w,c) = f(w,c;\theta)$.

Reward Function Based on Multi-source Ranking

However, the above solution of using the ranking model as the reward function is not optimal for the solution evaluation because it is unable to handle data from different sources. First, different features work in different sources. When ranking the candidates from CDG, the retrieval scores are dominant, while other features may be noisy. However, to rank candidates from vocabulary, the retrieval scores are useless. Second, one model cannot fit the data from different sources in a unified way since the information in these two sources is different. The data-driven approach is good at fine-tuning the ranking for a large number of samples, while it is hard for it to learn a global strategy for candidates from which source is more important. Thus, we design a novel reward function for matching the word s_i and the clue c_i as:

$$r(s_i, c_i) = \phi(s_i, c_i) + \omega_1 \rho(s_i, c_i) + \omega_2 I(s_i), \quad (9)$$

where ϕ and ρ are two ranking models for answers from the CDG and the vocabulary, respectively. The third item is a reward for the words in the vocabulary, $I(s_i) = max(S_{vocab}(s_i), S_{phrase}(s_i))$. Hyper-parameters $\omega_1 > 0$ and $\omega_2 > 0$ are used to control the reward scale of different sources. ϕ and ρ are both neural networks trained by data from different sources. ϕ is the neural network we trained for seen clue matching (i.e. S_{seen}). ρ uses other features except for S_{vocab} and S_{phrase}. I can be also regarded as a ranking model for out-of-vocabulary candidates. The training of ϕ has been mentioned before in the CDG. For the training of ρ, we construct a dataset by running a basic MCTS without the reward of ρ on the validation set and obtain the data samples from the candidate generation of the vocabulary source. After training ϕ and ρ, we tune the hyper-parameters ω_1 and ω_2 on the validation set based on the performance of MCTS.

Experiments

We conduct experiments to prove that our method is a state-of-the-art and that the designs in our method are effective[2].

Data Sources

We collect clues and puzzles from the web[3]. Apart from the puzzles and clues, we use a wide range of external sources such as Wikidata (Vrandečić and Krötzsch 2014) and several dictionaries (Miller 1998; Meyer and Gurevych 2012)[4]. The large vocabulary is a combination of the answers from past puzzles, dictionary words, as well as entity names and aliases from Wikidata.

Dataset for Crossword Puzzle Resolution

There are no existing publicly available standard datasets for CP problems. Previous methods (Littman et al. 2002; Ernandes et al. 2005) tested their method in their own selected puzzles, and the criteria for selection are not specified. So the difficulty of test puzzles is hard to compare. In addition, the source of the clue database also influences the difficulty of the test puzzle, because the more seen clues are in a CP, the easier it gets. Hence, it is an urge to construct a dataset with puzzles and clue database fairly selected.

We construct a dataset based on NYT daily crossword puzzles. To eliminate the influence of different clue databases, we select a split date, use the collection of clues before that date as the clue database, and use the puzzles after that date as the test set. According to this criterion, we construct a *standard test set* with 57 puzzles for test and 4,568,786 seen clues in the database. However, many puzzles are too simple and can be easily resolved because most of the clues can be found in the clue database, even though we set the split date. To show the capability of our algorithm, we select 24 puzzles with the lowest clue database recall rates as the *hard test set*. In addition, based on the same criterion, we construct a validation set with 85 puzzles for tuning hyper-parameters. The statistics of our datasets are presented in Table 2.

Category	Puzzles	Seen Clues
Validation	85	4,510,058
Test (Standard)	57	4,568,786
Test (Hard)	24	4,568,786

Table 2: Our crossword puzzle resolution dataset statistics.

Experimental Setup

To measure the performance of methods of crossword puzzle resolution, we evaluate the results using the ratio of correct words and correct letters based on the ground truth grid. Since the crossword puzzles are required to be finished in a limited time, we set two time limits (100 seconds and 1000 seconds) as different experiment settings for testing. To reduce the bias due to the randomness of MCTS, we run our algorithm 5 times and report the average performance.

We use grid search to tune hyper-parameters on the validation set. The values of hyper-parameters we use are $\lambda = 0.08$, $\tau = 0.2$, $\gamma_1 = 50$, $\gamma_2 = 200$, $\omega_1 = 0.7$ and $\omega_2 = 15$. For the model ρ, deeper models have better performance on the ranking but are inefficient to calculate. In practice, we adopt the linear models resulting in better performance of MCTS due to the efficient calculation during the search.

Search Algorithm Comparison

In this section, we compare our MCTS to other search algorithms for solving CPs. Since source codes of previous works on crossword puzzle resolution are not available, we re-implement the search algorithms but retain the candidate generation and scoring function of our method as baselines. The re-implemented baselines are only for the comparison of search method and might not be the best-performance system reported in their paper. The baselines are as follows.

- Webcrow (WA*) (Ernandes et al. 2005). Webcrow adopts a weighted A* search algorithm to do the grid filling. Our re-implemented version removes the web search module because it is time-consuming and might be a leakage for the test set because we only consider the seen clues before a certain date.
- WA*+Heuristic. We improve the WA* algorithm in Webcrow by using the heuristic from Dr. Fill as search order.
- Dr. Fill (Ginsberg 2011). It adopts LDS to solve the search problem in CP. Our re-implemented version does not have the feature of the crossword "merit" because it is manually constructed and not publicly available.

The experiment results are shown in Table 3. We can see that our method outperforms all the competitors on the dataset. In addition, a good heuristic can significantly improve the performance of A* search. In this experiment, we do not consider baselines such as (Moschitti et al. 2015) because they focus on ranking the candidates for individual clues. We will discuss their scoring strategy with the ranking model in the following experiments.

[2]Codes and datasets are publicly available at https://www.github.com/lhlclhl/CP.

[3]https://www.xwordinfo.com/;https://www.crosswordgiant.com/;http://www.otsys.com/clue

[4]We extract Wiktionary and Wikidata from 20210320 dumps.

Time Limits	100s		1000s	
Metrics	words	letters	words	letters
Webcrow (WA*)	36.21	47.16	36.21	47.16
WA*+Heurisitc	63.34	77.06	63.43	77.12
Dr. Fill (LDS)	88.04	93.99	89.69	95.17
Ours (MCTS)	**94.05**	**97.30**	**97.04**	**98.81**

Table 3: System comparison on the standard test set.

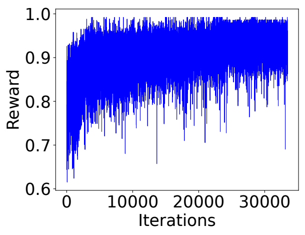
Study of MCTS

In this section, we conduct experiments to study the designs in our MCTS algorithm and how they influence performance. We use the *hard test set* to evaluate.

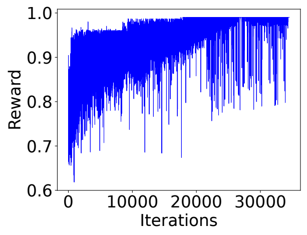
Effect of Variance Control for Tree Policy We propose a variance control technique for the UCT algorithm to relieve the problem of variant estimated Q-value in the CP problem. We conduct experiments to show the effectiveness of this special design. For comparison, we use the standard P-UCT algorithm for the tree policy (w/o VC) with re-tuned λ in the validation set, and other strategies, including setting the exploration factor to the action value (Balla and Fern 2009) ($\lambda = Q/V$) and the state value (Keller and Helmert 2013) ($\lambda = 1$). Also, we try other settings of λ to see how it affects the results ($\lambda = 0.01, 0.2, 0.5$). The results are presented in Table 4. We can see the performance drops without our variance control technique, which proves the effectiveness of our design. For the influence of the exploration factor, less exploration ($\lambda = 0.01$) encourages the algorithm to find better solutions in the short term, but limits its performance in the long run. Our setting ($\lambda = 0.08$ tuned on the validation set) can be regarded as a trade-off between the short-term exploitation and the long-term exploration. We further illustrate the effectiveness of the strategy by analyzing its runtime behavior during the search. We select a puzzle where the two versions of our method achieve the same results (word accuracy of 97.2%). We plot their reward-iteration curves in Figure 3. MCTS with our variance control strategy shows faster convergence and more steady improvement during the iteration.

Effect of Action Space Reduction In our MCTS algorithm, we design an action space reduction strategy where we limit the action space in those filling in the rank-1 candidate word for each clue, and an additional one of eliminating the rank-1 to consider the rest. This strategy is actually an exploitation and exploration trade-off. The strategy maximizing exploitation is to only consider the rank-1 candidates, and the one maximizing exploration is to consider all the candidate words at the same time. In this section, we conduct experiments to prove the effectiveness of this trade-off strategy by comparing it with those two alternatives.

- All Actions Included (AA). We consider filling all the candidate words as actions in MCTS. This is a strategy that emphasizes exploration.
- No Additional Action (NA). We only consider the rank-1 candidate for each clue without the additional action, which is inclined to the exploitation of the heuristic.



(a) w/o VC



(b) w/ VC

Figure 3: Reward-iteration curve comparison between the algorithm with and without variance control strategy.

Results in Table 4 show the effectiveness of the trade-off in our action space design. It is obviously a bad idea to consider all the candidates as actions. Although the NA strategy has slightly better performance in the short time limit, its lack of exploration limits the performance in the long run.

Study of Reward Function

In this section, we conduct experiments to explore how the reward function influences the performance of MCTS. We compare our design with different strategies for scoring CP solutions. Due to the different problem formalization, previous works on crossword resolution usually adopt a maximum log-likelihood objective to scoring the solution (Nicosia et al. 2015). Our reward function with similar features is empirically equal to the previous scoring strategy. The main difference here is how to obtain such a function to achieve the best performance of the search algorithm. We conduct experiments to show the superiority of our design by comparing it with previous methods to obtain the reward function. The baselines are as follows.

- Ranking model as reward (RMR). This is a commonly used strategy in some works (Barlacchi et al. 2014; Moschitti et al. 2015). For solution evaluation, it directly uses the output of the ranking model, which is trained by the mixed training samples of ϕ and ρ.
- Inverse Reinforcement Learning (IRL). Considering each solution as a trajectory in the MDP, this is an IRL problem. The objective of IRL is to obtain a reward function that gives high rewards for demonstration trajectories (good solutions in our case) and low rewards for others. We compare with the maximum margin planning (Ratliff et al. 2006) (MMP) algorithm. We run the IRL algorithm on the validation set with the initialized parameters of the ranking model.

The results are shown in the bottom part of Table 4. Our design of the reward function achieves the best performance over all the baselines. For the RMR baseline, we can see that the learned candidate ranking model serves as a bad reward function although it provides a more sophisticated ranking for each clue. In addition, the IRL baseline also shows inferior performance. The reason is that the negative solutions are usually undersampled and biased.

Time Limits	100s		1000s	
Metrics	words	letters	words	letters
Ours	91.20	96.15	94.88	**98.04**
w/o VC	90.20	95.53	92.89	96.99
$\lambda = Q/V$	88.61	94.75	94.21	97.75
$\lambda = 1$	87.04	93.52	93.38	97.25
$\lambda = 0.5$	89.06	94.50	94.15	97.45
$\lambda = 0.2$	90.31	95.73	**94.89**	97.99
$\lambda = 0.01$	**92.26**	**96.84**	93.76	97.35
AA	46.40	64.42	47.76	65.03
NA	91.31	96.22	94.15	97.36
RMR	85.70	92.88	89.20	95.00
IRL	85.15	92.02	87.98	93.54
w/o ϕ	6.18	16.50	8.54	18.90
w/o I	83.66	92.23	84.35	92.48
w/o ρ	87.92	94.80	89.65	95.67

Table 4: Ablation study of different designs in our MCTS algorithm on the hard test set.

For the ablation study, we eliminate factors in our reward function respectively and see how they influence the performance of MCTS. In Table 4, we can see that the most important part of our reward function is the score given by the clue-dependent retrieval ϕ. The performance of our method drops drastically without ϕ, but we don't think more analysis of ϕ is necessary because the information of ϕ is also included in comparative methods. If we remove features in ϕ for those methods, their performances drop drastically as well. The ranking model ρ is the least important, while it still brings performance gain of $3\% - 5\%$ on the word accuracy.

Error Analysis

There are two main types of errors. The first is the low recall rate of the candidate generation. The performance of our method is affected heavily by the confidence answers retrieved from the clue database. Due to the characteristics of crossword design, most of the clues (around 80% on average[5]) are textually similar to some seen clues in the database which means their answers can be retrieved from the clue database. When there are few seen clues or many clues are ambiguous, the performance of our method drops. Another factor leading to poor performance is caused by the imperfect reward function. In many cases, the reward function is not discriminative enough for good and bad solutions, which leads to poor performance of the algorithm.

Related Work

Search The traditional methods (Littman et al. 2002; Ernandes et al. 2005) for crossword puzzle resolution adopt the A* search (Pohl 1970) to find the best solution. The depth-first search suffers from the problem of "early mistakes" (Harvey and Ginsberg 1995). The algorithm is difficult to backtrack if the mistake happens in the shallow level of the search. To alleviate this problem, an LDS (Ginsberg 2011) is adopted based on a heuristic strategy. The LDS change the search order to first check the solutions that violate the heuristic strategy less. It is obvious that LDS heavily relies on the exploitation of the heuristic strategy and lacks the ability to explore other options. MCTS provides a trade-off between exploitation and exploration of this problem while has not been applied to this task. It has a profound impact on AI approaches for domains that can be represented as trees of sequential decisions, such as games (Silver et al. 2017; Raiko and Peltonen 2008), and some NLP problems (Liu et al. 2020).

Clue understanding Clues understanding includes generating candidates based on clues and evaluating the solution according to the clue-word matching. For the generation of candidates, traditional efforts tend to use complicated expert systems with many submodules (Littman et al. 2002; Ernandes et al. 2005). These heavy candidate generation modules are impractical, as CP problems usually have a time limit that the search and candidate generation share. Hence, the following work considers a few but effective sources and concentrates on the search to achieve better overall performance (Ginsberg 2011). With the development of natural language processing (Devlin et al. 2018) and knowledge base (Auer et al. 2007; Vrandečić and Krötzsch 2014), there are emerging solutions for similar tasks such as reverse dictionary (Zheng et al. 2020), entity retrieval (Gillick et al. 2019), question answering (Karpukhin et al. 2020), and cryptic clue decryption (Efrat et al. 2021; Rozner et al. 2021). However, it is still difficult for current methods to effectively and efficiently handle the ambiguity, obscurity, and diversity of the clues. For the evaluation of the matching between clues and words, traditional solutions use manually-tuned scoring functions to evaluate the clue-word matching (Ernandes et al. 2005; Ginsberg 2011), which has limited utility on big data. Some recent methods use the data-driven methods such as learning-to-rank models (Barlacchi et al. 2014; Moschitti et al. 2015; Severyn et al. 2015) to train the scoring function. However, they optimize the candidate ranking only for individual clues, and it is not optimal to be directly used to evaluate the whole solution because they do not consider grid constraints.

Conclusion

In this paper, we propose a novel framework for solving crossword puzzles based on MCTS. As far as we know, we are the first to apply MCTS to solve the crossword puzzle problem. We design a novel reward function combining the delicate candidate ranking for individual clues and robust global rewarding for the whole solutions. Our method achieves state-of-the-art performance in the dataset. There are three aspects of future works. The first is to design a more effective candidate generation module.The second is to design a more robust and discriminative reward function. The third is to introduce policy and value learning in MCTS.

[5]It does not mean 80% clues can be solved by the clue database because the retrieved answers are always mixed with noisy candidates. Thus the search still serves as a crucial role in the system.

Acknowledgements

We thank anonymous reviewers for their comments and suggestions. This work was supported by National Key Research and Development Project (No. 2020AAA0109302), Shanghai Science and Technology Innovation Action Plan (No. 19511120400), Shanghai Municipal Science and Technology Major Project (No. 2021SHZDZX0103), National Natural Science Foundation of China (No. 62102095), China Postdoctoral Science Foundation (No. 2020M681173 and No. 2021T140124), and National Science Foundation (No. NSF-DMS-2113637).

References

Auer, S.; et al. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*, 722–735. Springer.

Balla, R.-K.; and Fern, A. 2009. UCT for tactical assault planning in real-time strategy games. In *Twenty-First International Joint Conference on Artificial Intelligence*.

Barlacchi, G.; et al. 2014. Learning to Rank Answer Candidates for Automatic Resolution of Crossword Puzzles. In *CoNLL*.

Bonet, B.; and Geffner, H. 2012. Action selection for MDPs: Anytime AO* versus UCT. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 1749–1755.

Browne, C. B.; et al. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1): 1–43.

Czech, J.; et al. 2021. Improving AlphaZero Using Monte-Carlo Graph Search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, 103–111.

Devlin, J.; et al. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Efrat, A.; et al. 2021. Cryptonite: A Cryptic Crossword Benchmark for Extreme Ambiguity in Language. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 4186–4192.

Ernandes, M.; et al. 2005. Webcrow: A web-based system for crossword solving. In *AAAI*, 1412–1417.

Gillick, D.; et al. 2019. Learning Dense Representations for Entity Retrieval. In *Proceedings of the 23rd Conference on Computational Natural Language Learning*, 528–537.

Ginsberg, M. L. 2011. Dr. fill: Crosswords and an implemented solver for singly weighted csps. *Journal of Artificial Intelligence Research*, 42: 851–886.

Harvey, W. D.; and Ginsberg, M. L. 1995. Limited discrepancy search. In *IJCAI (1)*, 607–615.

Karpukhin, V.; et al. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 6769–6781. Online: Association for Computational Linguistics.

Keller, T.; and Helmert, M. 2013. Trial-based heuristic tree search for finite horizon MDPs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 23, 135–143.

Leurent, E.; and Maillard, O.-A. 2020. Monte-Carlo Graph Search: the Value of Merging Similar States. In *Asian Conference on Machine Learning*, 577–592. PMLR.

Littman, M. L.; et al. 2002. A probabilistic approach to solving crossword puzzles. *Artificial Intelligence*, 134(1-2): 23–55.

Liu, G.; et al. 2020. Extracting Knowledge from Web Text with Monte Carlo Tree Search. In *Proceedings of The Web Conference 2020*, 2585–2591.

Meyer, C. M.; and Gurevych, I. 2012. *Wiktionary: A new rival for expert-built lexicons? Exploring the possibilities of collaborative lexicography*. na.

Mikolov, T.; et al. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26: 3111–3119.

Miller, G. A. 1998. *WordNet: An electronic lexical database*. MIT press.

Moschitti, A.; et al. 2015. SACRY: Syntax-based Automatic Crossword puzzle Resolution sYstem. In *ACL*.

Nicosia, M.; et al. 2015. Learning to rank aggregated answers for crossword puzzles. In *European Conference on Information Retrieval*, 556–561. Springer.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4): 193–204.

Raiko, T.; and Peltonen, J. 2008. Application of UCT search to the connection games of Hex, Y,* Star, and Renkula! *AI and Machine Consciousness*.

Ratliff, N. D.; et al. 2006. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, 729–736.

Robertson, S.; and Zaragoza, H. 2009. *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc.

Rosin, C. D. 2011. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3): 203–230.

Rozner, J.; et al. 2021. Decrypting Cryptic Crosswords: Semantically Complex Wordplay Puzzles as a Target for NLP. *Advances in Neural Information Processing Systems*, 34.

Severyn, A.; et al. 2015. Distributional neural networks for automatic resolution of crossword puzzles. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, 199–204.

Silver, D.; et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359.

Vrandečić, D.; and Krötzsch, M. 2014. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10): 78–85.

Xia, F.; et al. 2008. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, 1192–1199.

Zheng, L.; et al. 2020. Multi-channel reverse dictionary model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 312–319.